Chaperones and Impersonators

Run-time Support for Reasonable Interposition

T. Stephen Strickland
Univ. of Maryland, College Park

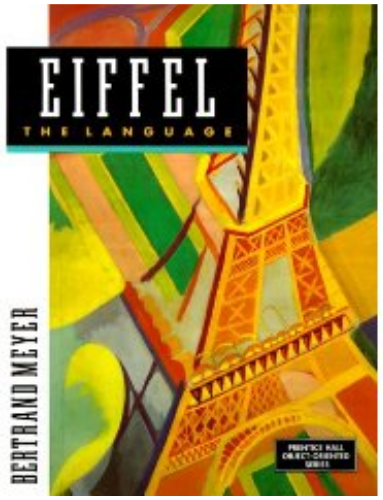
Sam Tobin-Hochstadt
Northeastern Univ.

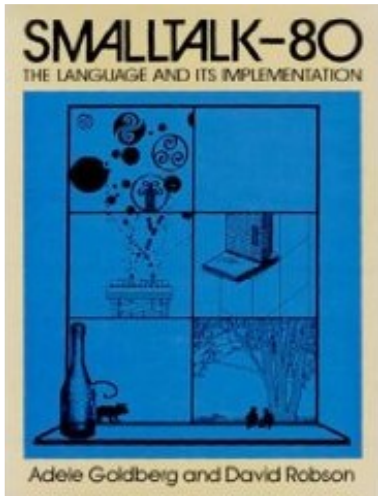
Robert Bruce Findler
Northwestern Univ.

Matthew Flatt
Univ. of Utah

OOPSLA, October 25, 2012

Contract Systems





Recent Uses of Contract Systems

Static Analysis

Cousot et al. An Abstract Interpretation Framework for Refactoring with Application to Extract Methods with Contracts. OOPSLA 2012.

Leino. Staged Program Development. OOPSLA 2012 Keynote.

Tobin-Hochstadt and Van Horn. Higher-order Symbolic Execution via Contracts. OOPSLA 2012.

Type Systems

Chugh et al. Dependent Types for JavaScript. OOPSLA 2012.

Takikawa et al. Gradual Typing for First-Class Classes. OOPSLA 2012.

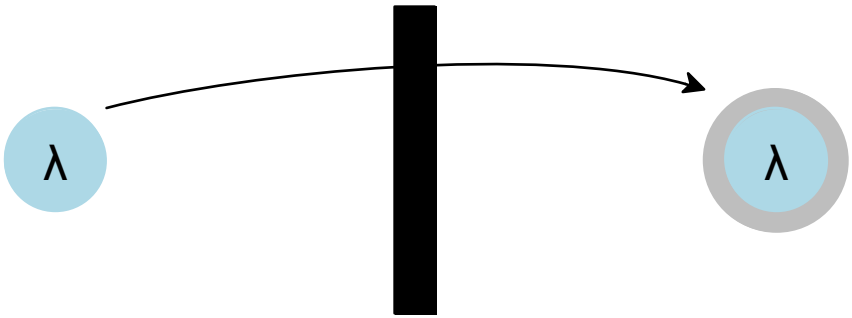
Higher-order Contracts

Contracts specified separately from values

Specifications may describe higher-order behavior

Contract system provides blame tracking

`server` `client`



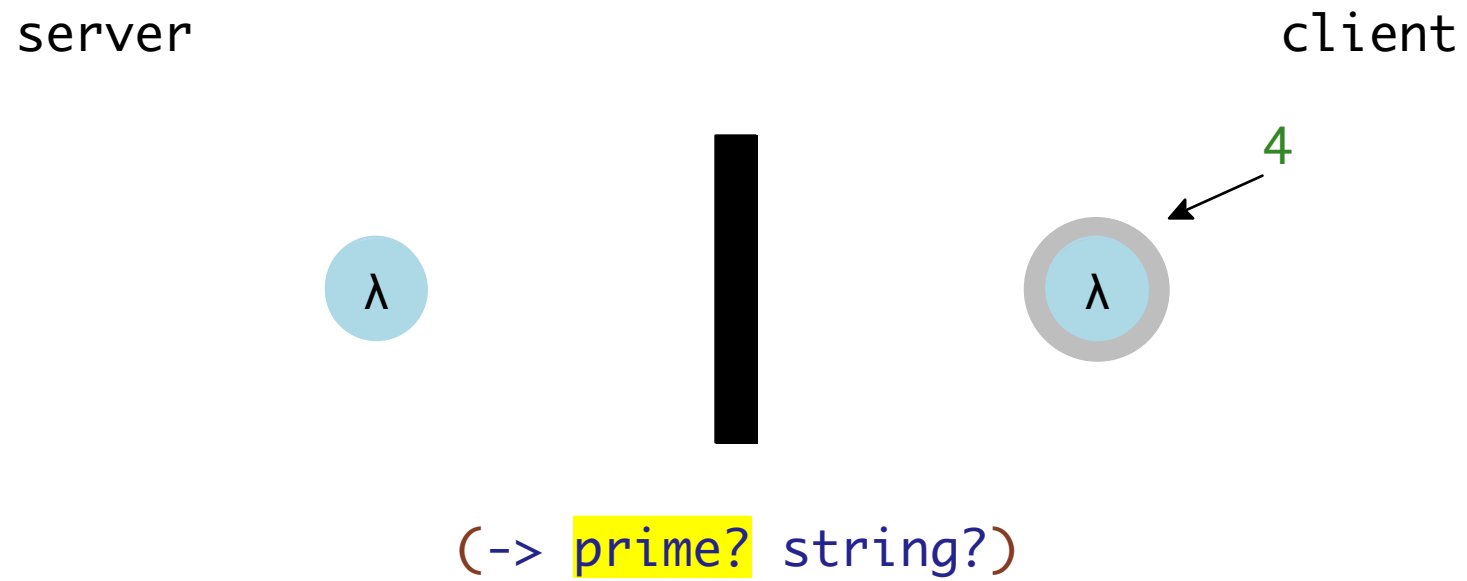

```
(-> prime? string?)
```

Higher-order Contracts

Contracts specified separately from values

Specifications may describe higher-order behavior

Contract system provides blame tracking



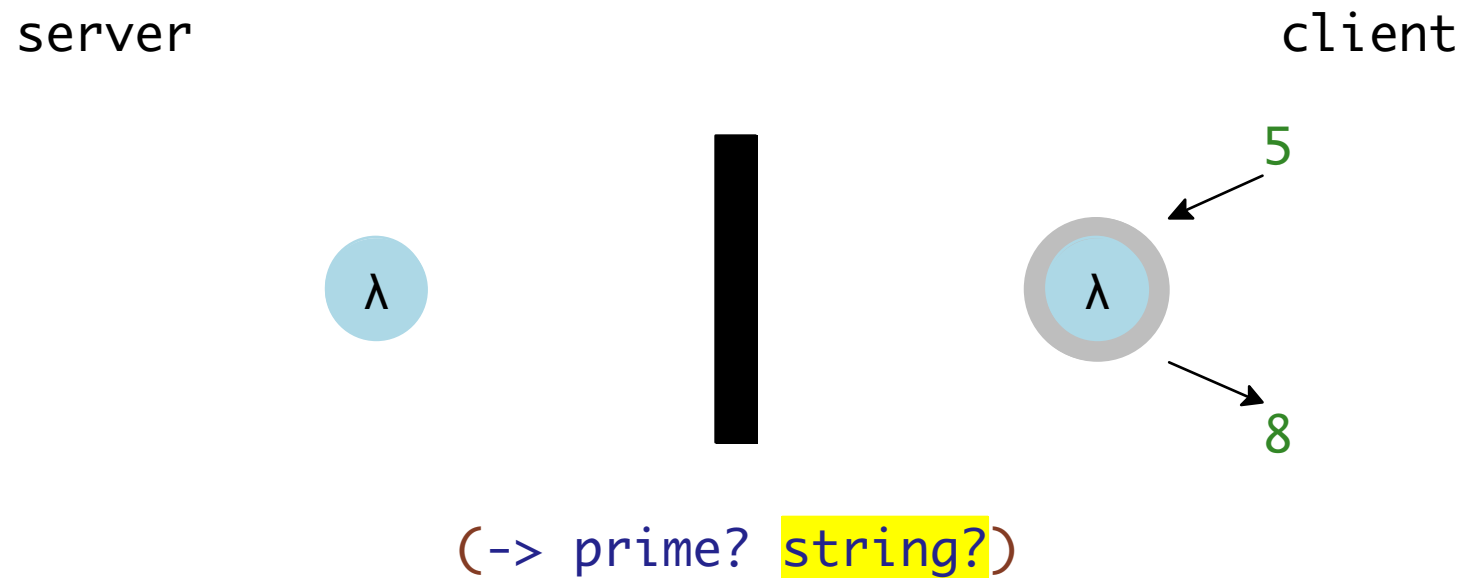
`client` broke the contract, expected `prime?`, got `4`

Higher-order Contracts

Contracts specified separately from values

Specifications may describe higher-order behavior

Contract system provides blame tracking



`server` broke the contract, expected `string?`, got `8`

Prior Support for Higher-order Contracts

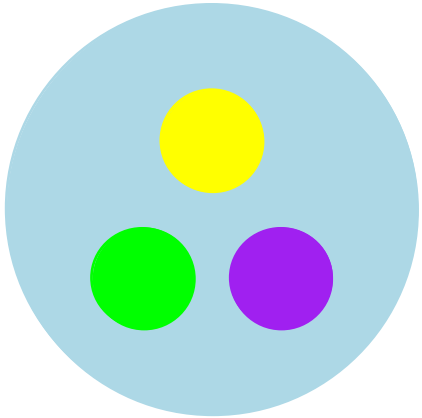
Functions	✓
Immutable containers	✓
Mutable containers	✗
Generative structures	✗

Prior Support for Higher-order Contracts

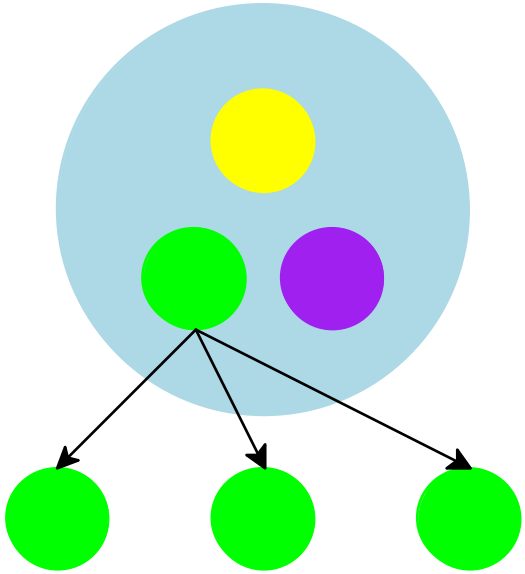
Functions	✓
Immutable containers	✓
Mutable containers	✗
Generative structures	✗

Let's use proxies!

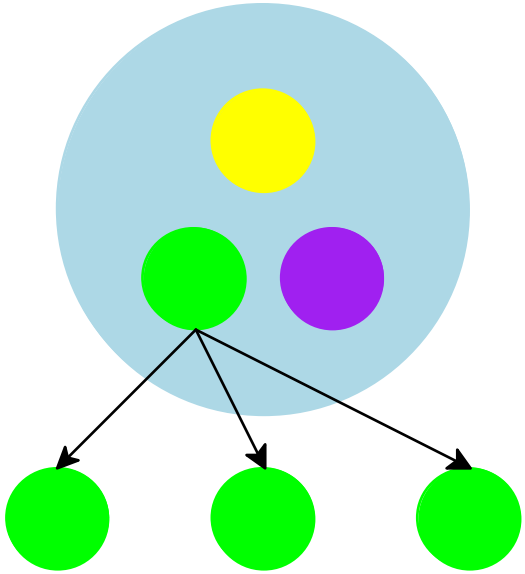
The Problem with Proxies

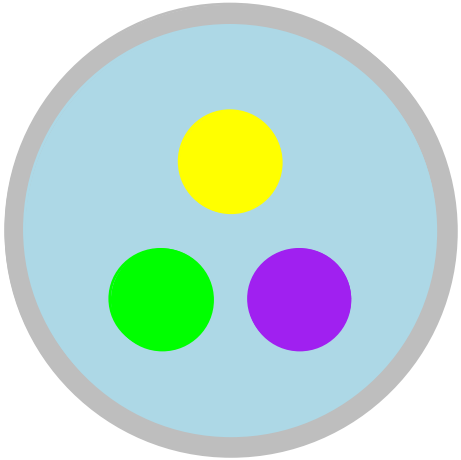


The Problem with Proxies

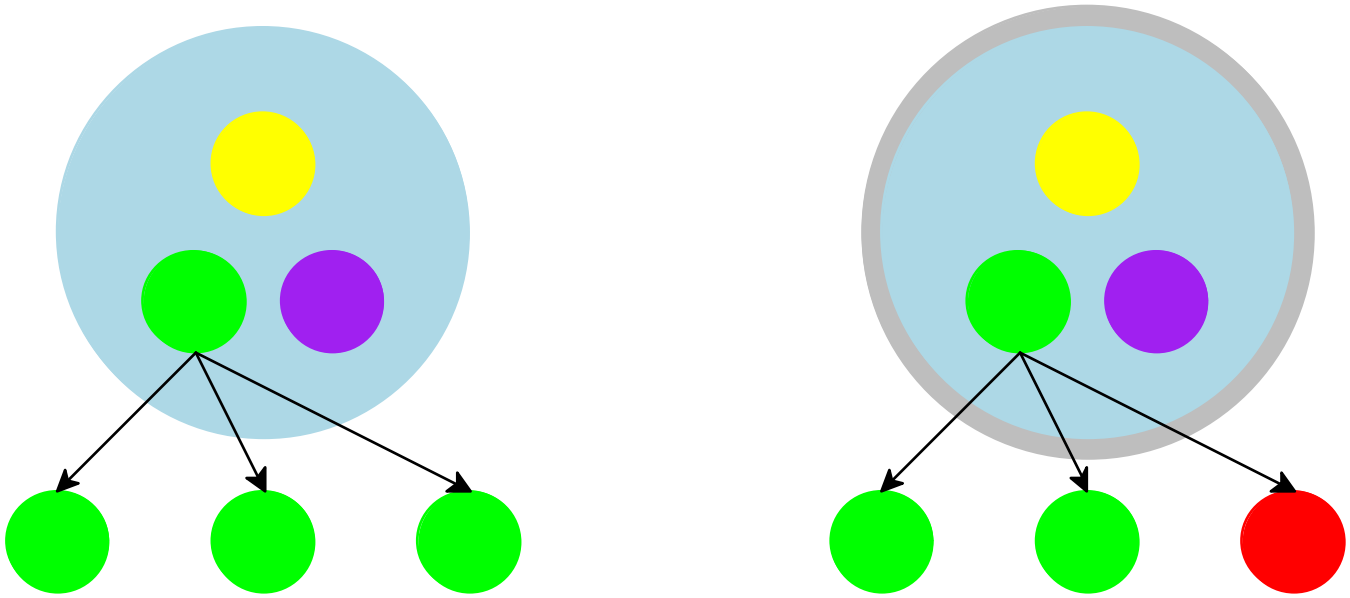


The Problem with Proxies





The Problem with Proxies



Our System of Proxies

Chaperones

- Restricted in changing behavior
- Applicable to more values

Impersonators

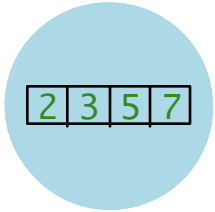
- Freer to change behavior
- Applicable to fewer values

Current Support for Higher-order Contracts

Functions	✓
Immutable containers	✓
Mutable containers	✓
Generative structures	✓

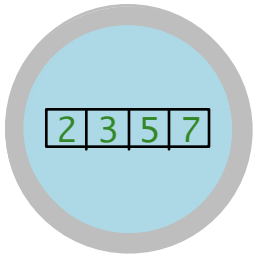
Vector Chaperones

```
(define vec1 (vector 2 3 5 7))
```



Vector Chaperones

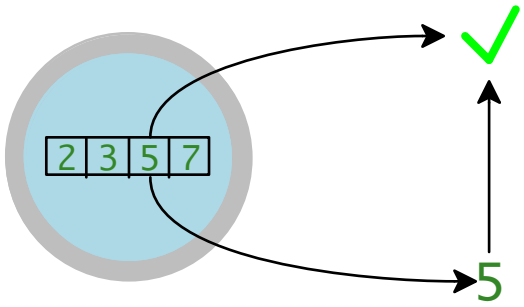
```
(define vec1 (vector 2 3 5 7))
(define vec2
  (chaperone-vector vec1
                    ))
```



Vector Chaperones

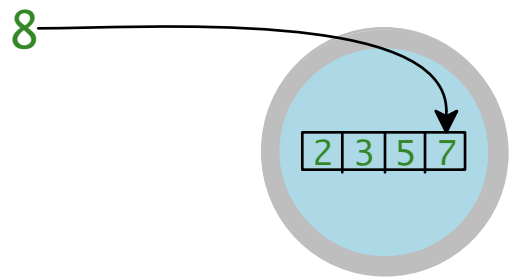
```
(define vec₁ (vector 2 3 5 7))
(define vec₂
  (chaperone-vector vec₁
   ; Interpose for vector-ref
   (λ (vec i v) (contract prime? v srv clt))

                                          ))
```



Vector Chaperones

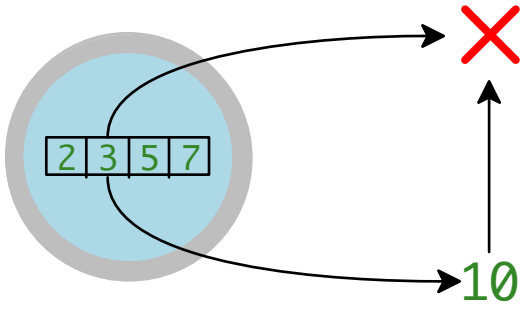
```
(define vec₁ (vector 2 3 5 7))
(define vec₂
  (chaperone-vector vec₁
   ; Interpose for vector-ref
   (λ (vec i v) (contract prime? v srv clt))
   ; Interpose for vector-set!
   (λ (vec i v) (contract prime? v clt srv))))
```



`clt` broke its contract, expected `prime?`, got `8`

Vector Chaperones

```
(define vec₁ (vector 2 3 5 7))
(define vec₂
  (chaperone-vector vec₁
   ; Interpose for vector-ref
   (λ (vec i v) 10)
   ; Interpose for vector-set!
   (λ (vec i v) (contract prime? v clt srv))))
```



non-chaperone result, original: 5, received: 10

Chaperone Restriction

Results of interposition functions must be a chaperone of the appropriate input.

(chaperone-of? v_1 v_2)

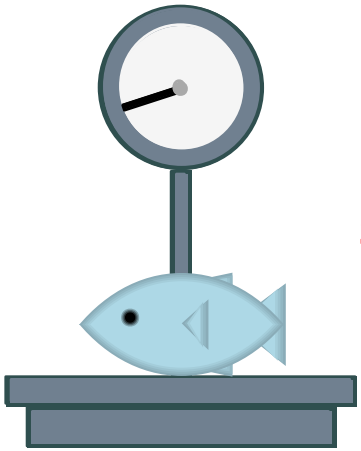
- If v_1 is equal to v_2, true.
- If v_1 is a chaperone of v_3, then check (chaperone-of? v_3 v_2).
- Otherwise, false.

Structure Chaperones

```
(struct fish (name weight))
(define f1 (fish "Dory" 14))
(define f2
  (chaperone-struct f1
   ; Operation to interpose
   fish-weight
   ; Interposing function
   (λ (s v) (contract prime? v srv clt))
   ...))
```

Structure Chaperones

```
(struct fish (name weight))
(define f1 (fish "Dory" 14))
(define f2
  (chaperone-struct f1
   ; Operation to interpose
   fish-weight
   ; Interposing function
   (λ (s v) (contract prime? v srv clt))
   ...))
```



srv broke its contract, expected `prime?`, got `14`

Chaperone Limitations

Inputs and results of operations must behave like originals.

Sealing contracts

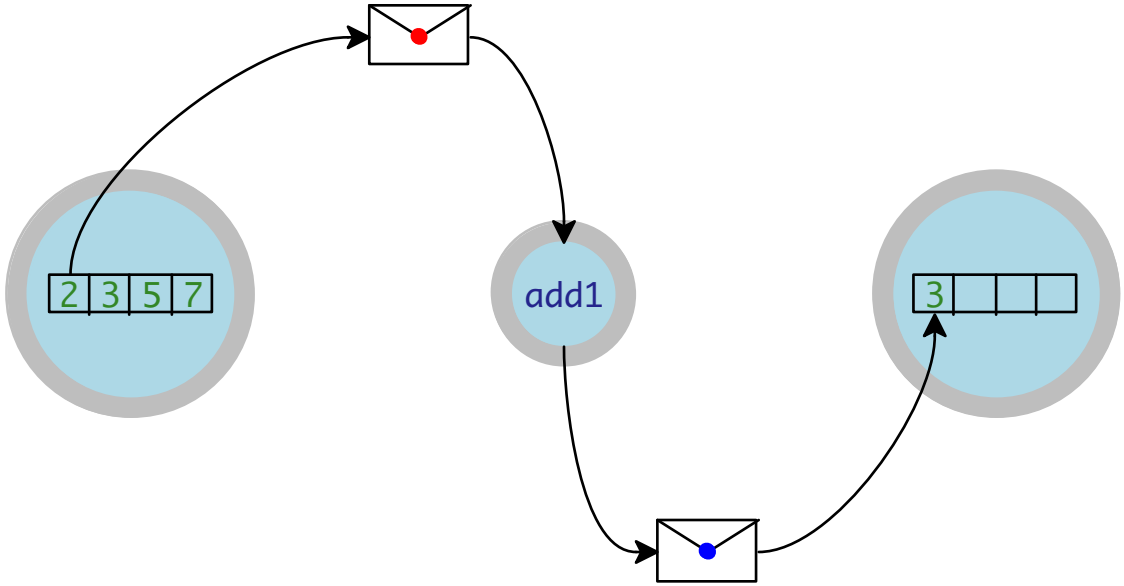
Guha et al. Relationally-Parametric Polymorphic Contracts. DLS 2007.

Matthews and Ahmed. Parametric polymorphism through run-time sealing. ESOP 2008.

Takikawa et al. Gradual Typing for First-Class Classes. OOPSLA 2012.

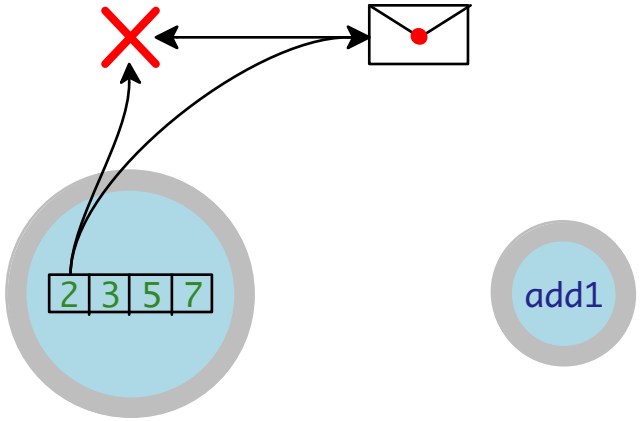
Chaperone Limitations

```
vector-map: (∀/c [A B] ((A . -> . B)
                        (Vectorof A) . -> .
                        (Vectorof B)))
```



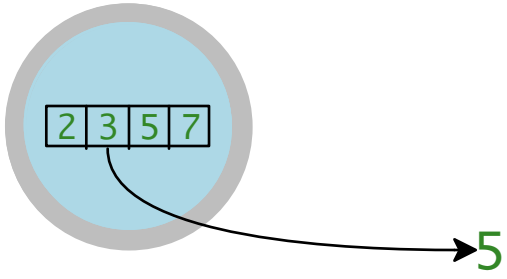
Chaperone Limitations

```
vector-map: (∀/c [A B] ((A . -> . B)
                        (Vectorof A) . -> .
                        (Vectorof B)))
```



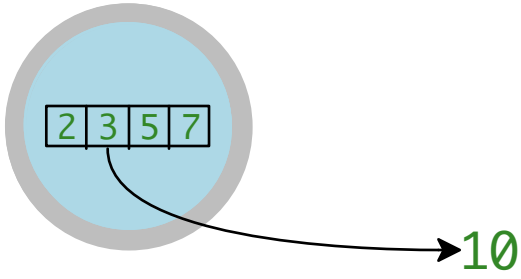
Vector Impersonators

```
(define vec1 (vector 2 3 5 7))
(define vec2
  (impersonate-vector vec1
   ; Interpose for vector-ref
   (λ (vec i v) (contract prime? v srv clt))
   ; Interpose for vector-set!
   (λ (vec i v) (contract prime? v clt srv))))
```



Vector Impersonators

```
(define vec₁ (vector 2 3 5 7))
(define vec₂
  (impersonate-vector vec₁
   ; Interpose for vector-ref
   (λ (vec i v) 10)
   ; Interpose for vector-set!
   (λ (vec i v) (contract prime? v clt srv))))
```



Impersonator Restrictions

No impersonators for immutable containers.

No impersonators for immutable fields of generative structures.

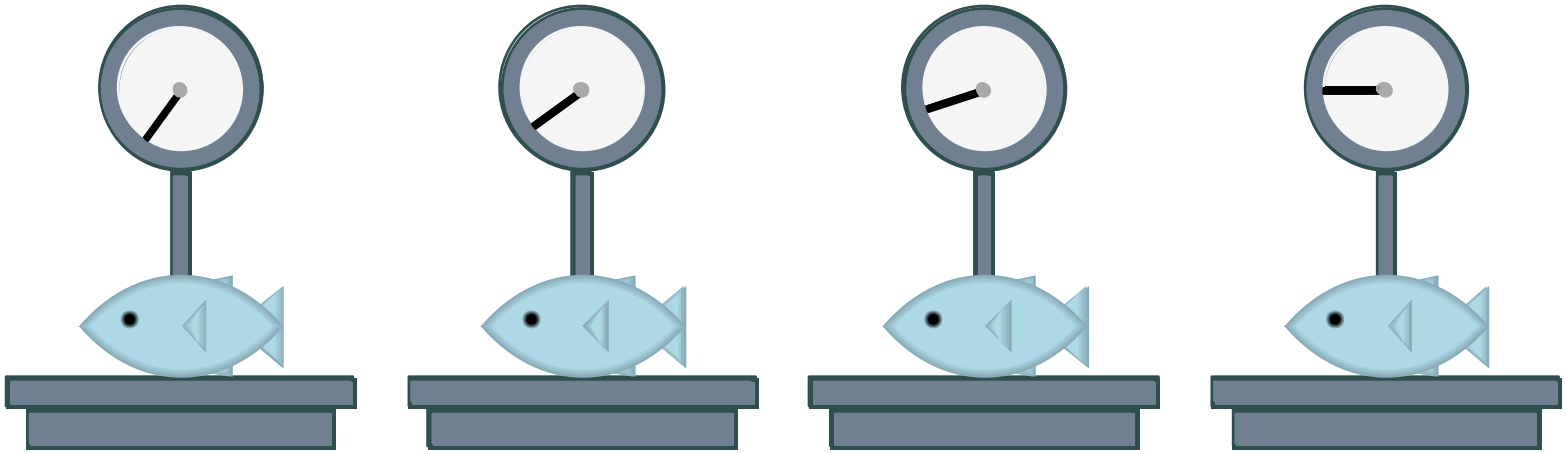
```
(struct fish (name weight))
(define f1 (fish "Dory" 12))
(define f2
  (let ([counter (fish-weight f1)])
    (impersonate-struct f1
     fish-weight
     (λ (f v)
       (begin0 counter (set! counter 1))))))
```

Impersonator Restrictions

No impersonators for immutable containers.

No impersonators for immutable fields of generative structures.

```
(struct fish (name weight))
(define f1 (fish "Dory" 12))
(define f2
  (let ([counter (fish-weight f1)])
    (impersonate-struct f1
     fish-weight
     (λ (f v)
       (begin0 counter (set! counter 1))))))
```



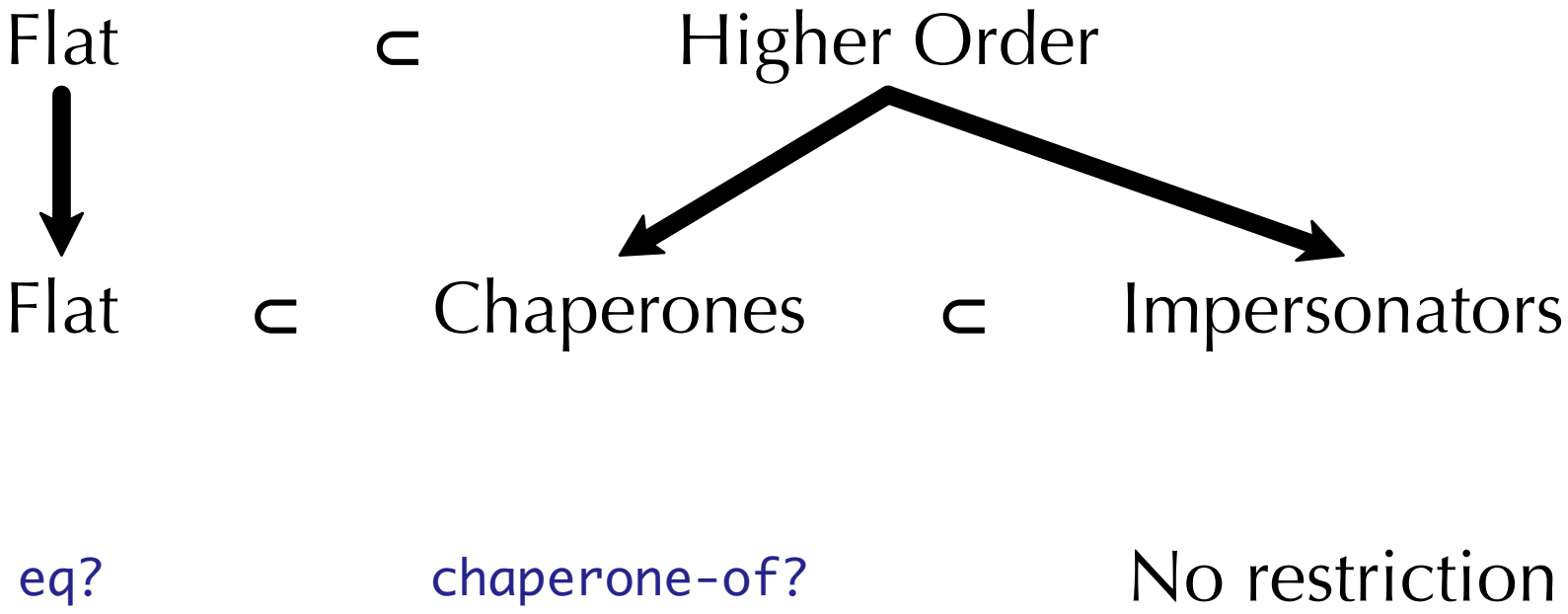
Impersonator Restrictions

No impersonators for immutable containers.

No impersonators for immutable fields of generative structures.

```
(struct fish (name weight))
(define f1 (fish "Dory" 12))
(define f2
  (let ([counter (fish-weight f1)])
    (impersonate-struct f1
     fish-weight
     (λ (f v)
       (begin0 counter (set! counter 1))))))
```

Contract Hierarchy



A General Proxying System

Revocable Membranes

Mark Miller. Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. PhD Thesis.

Local views on remote data

SMTP server access that appears like local hash tables and vectors

Performance (in seconds)

benchmark	no proxy	no checks	impersonate	chaperone
make guide	10.467	10.606	10.818	10.792
render guide	1.889	2.044	3.741	3.727
keyboard	5.182	5.231	7.253	7.258
slideshow	4.663	4.776	5.168	5.180
plot	1.854	1.886	2.362	2.394
typecheck	22.610	24.144	47.302	47.816
ode-apply	7.794	9.265	10.236	10.632

Performance (in seconds)

benchmark	no proxy	no checks	impersonate	chaperone	overhead
make guide	10.467	10.606	10.818	10.792	0%
render guide	1.889	2.044	3.741	3.727	0%
keyboard	5.182	5.231	7.253	7.258	0%
slideshow	4.663	4.776	5.168	5.180	0%
plot	1.854	1.886	2.362	2.394	1%
typecheck	22.610	24.144	47.302	47.816	1%
ode-apply	7.794	9.265	10.236	10.632	4%

Conclusion

Unrestricted proxies break programmer and compiler invariants.

Providing restricted proxies avoids this issue.

We now provide both via chaperones and impersonators in Racket.

http://racket-lang.org/